Machine Learning Engineer Nanodegree

Capstone Project

Abd Elrahman adel elsayied

12 April, 2019

I. Definition

Project Overview

AI (Artificial Intelligence) and machine learning is an important fields that helps in solving many problems like computer vision and self-driving cars. Recognizing objects in images and classifying it is an important task because it helps organizations to fasten their work to save its time.

In this project I have a deep neural network that would be capable of classifying 40 distinct face. This project could help in the security domain for work spaces monitoring systems, attendance monitoring system for students and employees.

Referances:

- Face Recognition-based Lecture Attendance System

Problem Statement:

In paper systems it take a lot of time to record the attendance of students or employ. So I want to make a deep learning model that can identify every person (student, employ) in the place.

I have got the dataset from kaggle has 400 labeled images. I want to make a classifier that can take an image and predict person identity.

The tasks involved are the following:

- Step 0: Download the data set.
- Step 1: take a subset of the training data set to be used as validation set.
- Step 2: Import Datasets.
- Step 3: preprocessing steps supply images to a pre-trained network in Keras.
- Step 4: Extract Bottleneck Features for Train set, valid set, Test Set.
- Step 5: rescale the images .
- Step 6: Obtain Bottleneck Features.
- Step 7: create Model Architecture.
- Step 8: Train the Model.
- Step 9: Test the Model.

Metrics

I evaluated my model using accuracy score test on the test set to check the accuracy of my model.

Accuracy = ($true\ positives$ + $true\ negatives$) / $dataset\ size$

This metric was used when evaluating the classifier because there is no imbalance in the data.

II. Analysis

Data Exploration

For the project I have used use a dataset I have got from kaggle which contains 400 face images 40 distinct person.

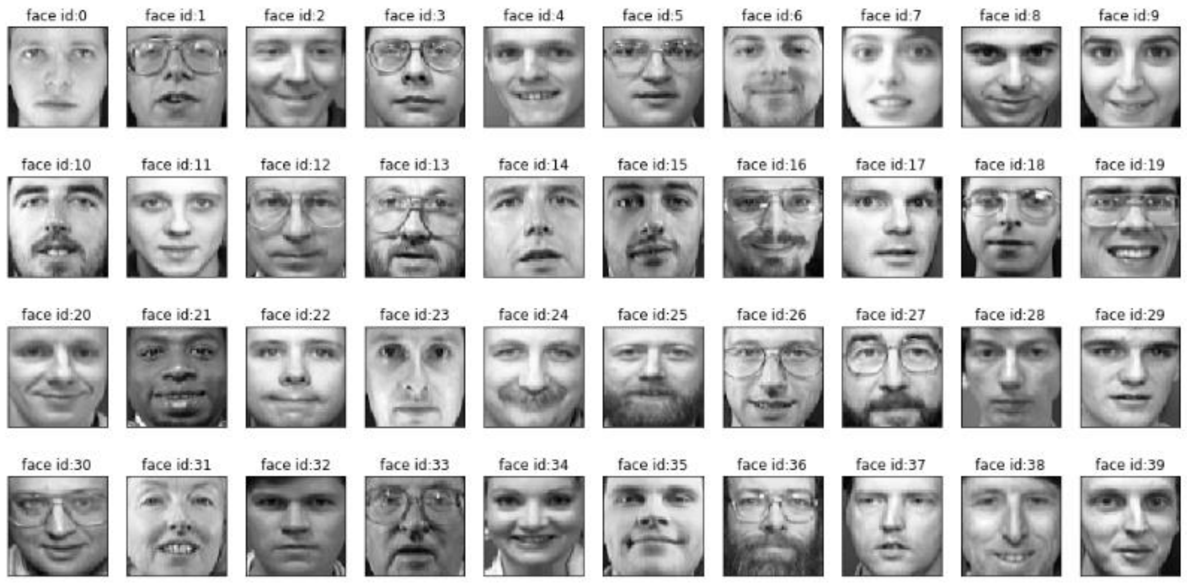
The content of dataset:

- Total number of the 40’s person images: 400.
- Train set: 280.
- Valid set: 80.
- Test set: 40.
- Number of classes: 40 ￼ Image size: The original dataset consisted of 92 x 112, while I reshape it to be 299 x299

I will split the training set into training and validation set to maintain the balance of the dataset.

The data contains 6 images for each person as a training set (total=240) and 4 as a test for each type (total=160).

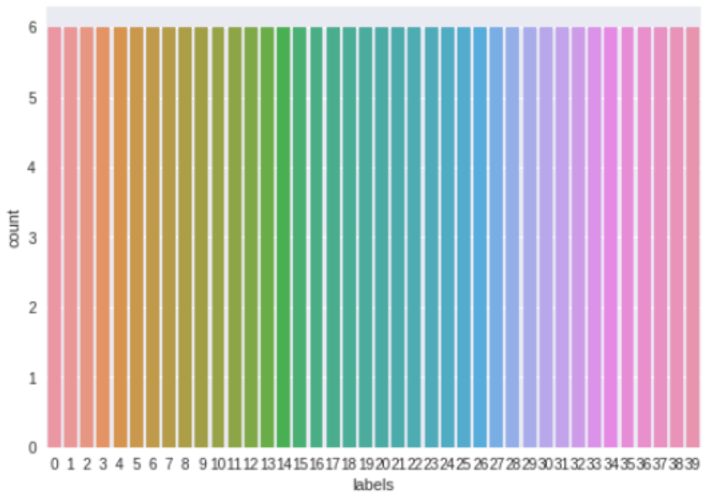
A preview for the 40 distinct faces in database:



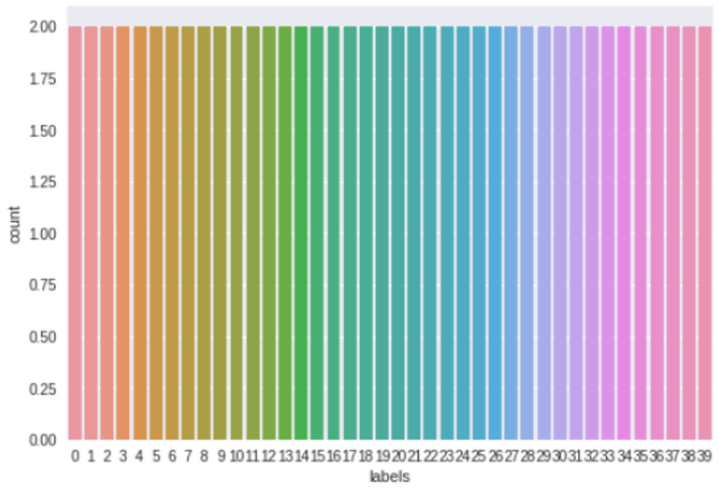
Exploratory Visualization

The figures below show subsets of the dataset and as we can see that each type has approximately an equal number of images.

1. **Train data Visualization (6 Images for train data)**



2. **Test and validation data Visualization (2 images for each of test and validation data)**



Algorithms and Techniques

The classifier is convolutional neural networks which is the main algorithm in for most image processing problems that are solved with machine learning. It needs a large amount of training data compared to other approaches. The algorithm outputs an assigned probability for each class. This is very good because using a classification threshold, the number of false positives can be reduced.

- The tradeoff is this increases the number of false positives.

The following parameters can be tuned to optimize the classifier:

- The classification threshold (as mentioned above).
- Solver (what algorithm to use for Transfer learning).
- Training length (number of epochs).
- Batch size (how many images to look at once during a single training step).
- Neural Network Architecture.
- Number of Layers
- Layer Types (Convolutional, fully connected or pooling)

I used Transfer Learning which focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For these types of problems, it is common to use a deep learning model pre-trained for a large and challenging image classification task such as the ImageNet 1000-class photograph classification competition. In my case InceptionV3 model was used.

Input:

When using Tensor Flow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape (nb_samples, rows, columns, channels)
Where nb_samples corresponds to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.

Benchmark Model:

I used this model to benchmark my model .

Model's structure:

```
model = Sequential()
```

```
model.add(Conv2D(filters = 20, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (64,64,1)))

model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 50, kernel_size = (6,6),padding = 'Same',
                 activation ='relu'))

model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 150, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (64,64,1)))

model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(40, activation = "softmax"))
```

with accuracy 97.5%, I'm looking to approach to this accuracy.

III. Methodology

Data Preprocessing

The preprocessing done in the "preprocessing steps supply images to a pertained network in Keras" notebook consists of the following steps:

The path_to_tensor function takes a gray image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image ,convert it to RGP and resizes it to a square image that is 299×299 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor.

In this case, since I am working with color images, each image has three channels. Likewise, since I am processing a single image (or sample), the returned tensor will always have shape (1, 299, 299, 3)

The paths_to_tensor function takes a numpy array image as input and returns a 4D tensor with shape (nb_samples, 299, 299, 3)

Here, nb_samples is the number of samples, or number of images, in the supplied array of image paths. It is best to think of nb_samples as the number of 3D tensors (where each 3D tensor corresponds to a different image) in my dataset!

Implementation

The implementation process can be split into two main stages:

- Extract Bottleneck Features for Train set, valid set, and Test Set stage.
- The classifier training stage.

During the first stage Bottleneck Features for Train set, valid set, and Test Set Extracted from the data. This was done in a Jupiter notebook (titled “Face Recognition”), and can be further divided into the following steps:

1. Load images into memory, preprocessing them as described in the previous section.

2. Load model that used for Transfer learning.

3. Check if the Bottleneck Features file is exist or not.

4. If the file is not exist calculate the features of the data by predict the preprocessed data.

5. Save the features to .npz file.

The second stage can be further divided into the following steps:

1. Obtain Bottleneck Features

Load the features from .npz files to the memory.

2. Create Model Architecture:

I used a standard Architecture that contain of two layers:

a. GlobalAveragePooling2D

Take the training features shape as input shape.

b. Dense

Hidden layer with 180 nodes.

The output layer with 40 node.

c. Dropout

Applies Dropout to the input with 0.3 rate.

3. Define the loss function, accuracy.

4. Train the network, logging the validation/training loss and the validation accuracy.

5. If the accuracy is not high enough, return to stage 1 and choose another model for transfer learning.

6. Save and freeze the trained network.

The coding process went smoothly, except the part of training the network it took me a long time and search to have a good model that could get a good validation loss.

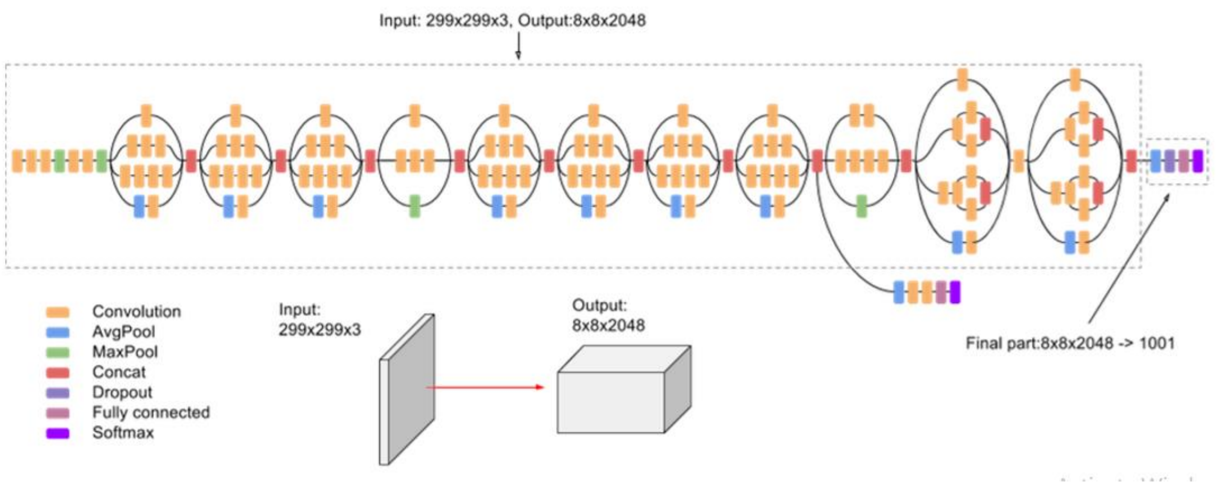
Refinement

In fact, there was a small improvement process as mentioned in the implementation section in the first phase. I had to try more than one model to use in the transfer learning process and make the most of the technique to improve accuracy. My last choice to use InceptionV3 instead of models like VGG-19, ResNet-50 and Xception was no surprise to me because it is the most popular pertained model comparing it to the others and have been used a lot in similar problems

IV. Results

Model Evaluation and Validation

During development, a validation set was used to evaluate the model. The final architecture and hyper parameters were chosen because they performed well. The structure of the model used for transfer learning:



My final model architecture:

```
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2 ( (None, 2048)              0

_________________________________________________________________
dense_3 (Dense)              (None, 180)               368820

_________________________________________________________________
dropout_2 (Dropout)          (None, 180)               0

_________________________________________________________________
dense_4 (Dense)              (None, 40)                7240
=================================================================
Total params: 376,060
Trainable params: 376,060
Non-trainable params: 0
```

My Final test accuracy is 96.25%

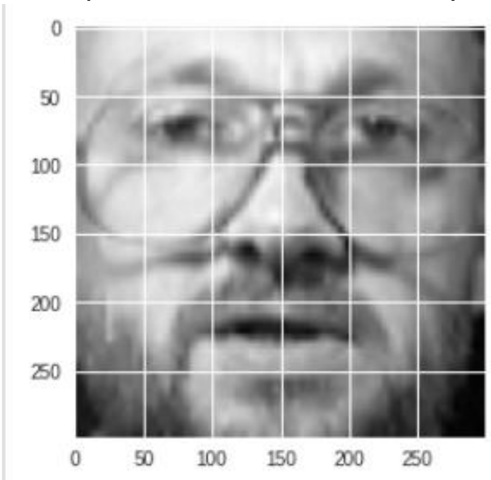
Justification

My Final test accuracy is 96.25% which close to my benchmark model accuracy. We can see that my model is well enough in classifying the these 40 people's faces And it will fit in security systems with little optimization.

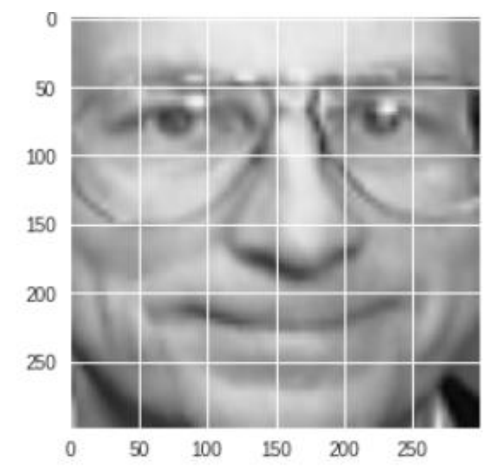
V. Conclusion

Free-Form Visualization

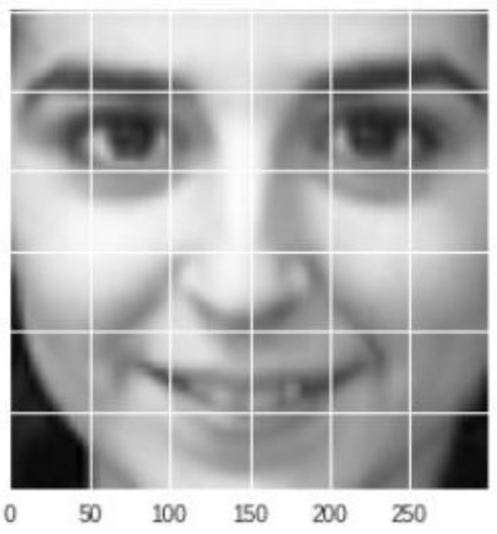
Examples of faces classified by the classifier.



```
That's a person. NO: 1
```



```
That's a person. NO: 3
```



```
I can't detect anything!
```

Reflection

The process used for this project can be summarized using the following steps:
1. An initial problem and relevant, public datasets were found.
2. The data was downloaded and split to training, test and validation subsets.
3. A benchmark was found.
4. The data was preprocessed.
5. Extract Bottleneck Features for Train set, valid set, and Test Set.
6. The classifier was trained using transfer learning technique on the data.
7. The classifier was tested.

I found steps 1 and 5 the most difficult, as I was working with a data set collected by me but I had a lot of troubles with it so I decided to work with Olivetti dataset for now until I construct a well formed one. And training the classifier that I was not familiar with before the project.

As for the most interesting aspects of the project, I'm very glad that I found the faces data set, as I'm sure they'll be useful for later projects/experiments. I'm also happy about getting to use transfer learning technique, as I believe it will be very useful in the future.

Improvement

As mentioned earlier there was some failure cases but to solve the bigger problem. I think that dataset have to be expanded with more Images

As a future work I will replace Olivetti dataset with manually collected dataset (By me) and integrate this code as apart of bigger system (attendance monitoring system)

Also I will use face recognition library to extract faces form images before it pass to the classifier.